# Design of a transputer network for searching neighbours in M.D. simulations

Bekker, H.; Renardus, M.

*Published in:*
Default journal



*Publication date:*
1990

North-Holland
Microprocessing and Microprogramming 30 (1990) 159-166



# Design of a transputer network for searching neighbours in M.D. simulations

H. Bekker and M. Renardus
Department of Mathematics and Computing Science, University of Groningen
P.O.B. 800, 9700 AV Groningen, THE NETHERLANDS
This project is part of FOM/STW project GCH88.1602

**Keywords:***Molecular Dynamics, Transputers, Neighbour Searching.*

## 1.1 Introduction

Molecular dynamics (M.D.) simulation is a widely used computational technique to study many body (atom) systems and their properties. In essence, in an M.D. simulation Newtons law is numerically integrated over many time steps for every particle in the system. In an N particle system the force on a particle is in principle calculated as a sum of forces exerted by N-1 other particles. This net force is used to calculate the new speed and this speed is used to calculate the new position of the particle. Repeating this procedure gives the time development of the micro state of the system.

With infinite computer power at hand, this informal description should be enough to realize a general purpose M.D. simulation program. However, the far too limited performance of available computers, including modern supercomputers, restricts the number of particles in the simulated system to some tens of thousands, and restricts the simulated time span to at most a nanosecond (which is approx. $10^6$ timesteps). Even with these strongly restricted systems, much attention has to be paid to spend only computer power on relevant parts of the M.D. algorithm. One of the most important techniques to reduce the number of forces to be calculated on particle $i$, is to consider only particles within a distance $R_{cutoff}$ ($R_{co}$). The force of particles outside the range $R_{co}$ is supposed to be zero. The neighbour list of particle $i$ is a list of particle numbers starting with $i$ and followed by the numbers of all the particles within cutoff range of $i$.

Until now, M.D. simulation programs run on general purpose computers ranging from PC's to modern supercomputers. Many interesting results have been obtained in this way but the performance of existing machines limits the simulation of more complex systems over a longer time span. In the past, some special purpose M.D. machines have been constructed [1] but the class of solvable M.D. problems on these machines is too narrow to be of interest for future M.D. simulations on complex chemical and biological systems. For this reason, at our department we are designing and constructing a special purpose M.D. computer with a tenfold performance relative to existing supercomputers. From existing M.D. simulation programs, running on general purpose computers, we learned that neighbour searching and force calculation together consume more than 90% of the total computer time. That is why our computer will consist of three main parts (fig. 1): a neighbour searching transputer network (NS) consisting of approx. 108 T222 transputers, a pipeline processor to do the actual force calculation on atom pairs (Non Bonded Force Processor: NBFP) consisting of approx. 80 floating point ALU's, and a transputer network to calculate special forces, new velocities and new positions after every timestep (Bonded Force and Update Processor: BFUP) consisting of approx. 64 T800 transputers.

To get enough bandwidth between these parts, communication takes place over transputer link bundles. Between the BFUP and NS are eight 20Mbits/sec links, between the NS and NBFP are twelve 20Mbits/sec links.

In the sections to follow we will concentrate on the NS part of our M.D. computer.

## 1.2 Neighbour searcher specification

As explained earlier the NS is connected to the BFUP with eight links. After M timesteps (M is typically 10) the BFUP sends the positions of a subset of all the particles in the system to the NS. We will call these particles 'indicator particles'. This data is send in packages with the format: a 16 bit particle number and a 48 bit (3*16 bit) particle position. For this transmission, all eight links are used simultaneously, packages have no link preference and arrive unordered. After arrival of the last package, the NS starts calculating and outputting neighbour lists. A neighbour list consists of

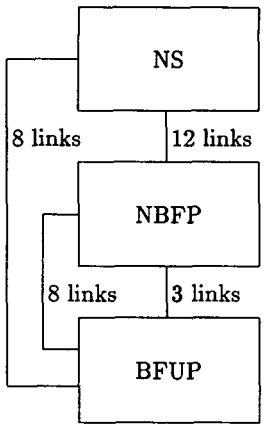


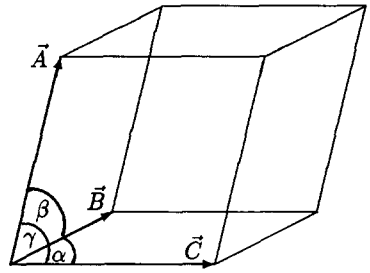

Figure 1: M.D. simulation computer overview.

records with the format: 16 bit particle number, 8 bit extension. The number of records in a neighbour list ranges from 2 to 64K, but is typically 100. Neighbour lists are transmitted to the NBFP via twelve 20 Mbits/sec links without link preference, however a neighbour list should not be distributed over two or more links. The performance of the NS should be high enough to send, on the average, every 200nsec a 24 bit neighbour list record to the NBFP.

A naïve specification of what the NS has to do could be 'Find for every incoming indicator particle its neighbours within distance $R_{co}$'. However, things are a bit more complicated than that by four factors: periodicity of the M.D. system, expansion, extension and exclusion. We will briefly explain these four notions.

(To appreciate the topology of the NS network, only the part about systems with periodicity has to be read. Expansion, extension and exclusion are described for the sake of completeness.)

**Systems with periodicity.** Some M.D. simulations are done on isolated clusters of particles. However, very often one wants the system to behave crystal like, that is, as if it is surrounded in every direction with replica systems. We will call the original system the 'primitive system', and surrounding systems 'image systems'. For single systems it is not necessary to define system borders but for systems with periodicity it is, because such systems have to be stacked together in a space filling way. The most general shape, capable of filling the whole space with only copy-and-translate actions, is the triclinic shape (fig. 2). The triclinic shape is defined by three non coplanar vectors $\vec{A}, \vec{B}, \vec{C}$. In order to fill the whole space, only translations of the form $k\vec{A} + l\vec{B} + m\vec{C}$, k,l,m:integer, are allowed. Conceptually, neighbour searching can now be described as:




$0 < \alpha \leq \beta \leq \gamma < 180$

Figure 2: Triclinic primitive system.

input (from the BFUP) all positions of particles in the primitive system, surround the system with an infinite number of image systems and calculate the neighbour list for every particle in the primitive system. Because $R_{co}$ never exceeds $1/2 \cdot \max(|\vec{A}|, |\vec{B}|, |\vec{C}|)$, only one layer of surrounding image systems is needed or, even better, only a layer of $R_{co}$ is needed.

**Expansion.** The incoming particles (indicator particles) from the BFUP do not represent single particles. Every particle stands for one to five particles, i.e. an indicator particle *ind-part-j* can be expanded to a set of particles expand(*ind-part-j*). The set of particles expand(*ind-part-j*) does not contain indicator particles except for possibly *ind-part-j*. (*ind-part-j* in this set can not be expanded recursively) Before an M.D. simulation run, the NS is loaded with the expansion set of every indicator particle. Expansion sets do not change during a simulation.

Neighbour searching should be done on the set of indicator particles. This means that, for all *ind-part-j*, if *ind-part-j* is within range $R_{co}$ of *ind-part-j*, then for every particle in the set expand(*ind-part-j*) a neighbour list should be made containing the particles expand(*ind-part-j*).

**Extension field.** Particle numbers in neighbour lists, produced by the process of searching, but not yet expanded, should get an extension. This extension is used to tell the NBFP which system this particle comes from, e.g. the primitive system or one of the image systems. Because only image particles from the first layer of image systems are within $R_{co}$ of particles in the primitive system, the numbers 0..26 suffice to number systems. 0 for the primitive system, 1..26 for image systems. The extension field should also indicate the first particle of a neighbour list. In the process of expansion, particles inherit the extension of the particle they are expanded from.

**Exclusion.** The neighbour list of every particle, so not only indicator particles, should undergo the process of exclusion. This means that the neighbour list should

be checked on particles to be excluded from it. The exclusion set of every particle is loaded into the NS before an M.D. run, and does not change. The number of particles in the exclusion list of a particle is typically four.

The neighbour lists produced after possibly creating image systems, searching, extension, expansion and exclusion can now be sent to the NBFP.

**Design freedom.** The specification as given is an infinite precision specification. However, an important design freedom is in the required smoothness of the cutoff sphere; the cutoff sphere does not have to be smooth at all, eight bits are enough for the sphere diameter $R_{co}$. This makes it possible to use 16 bit integers for the particle positions, so the T222 transputer is a suitable building block for the NS. (The T222 is a 16 bit transputer with four links, 4Kbyte on chip memory and an address space of 64K. Price $ 20.)

**Literature.** Some literature exists about neighbour searching in M.D. simulations, of which [9] gives an overview. The methods described are of two types: one type generates lists of neighbour particles which are updated only so many timesteps [2..4], and the other makes use of grid techniques for finding neighbours [5..7]. These methods have been implemented on general purpose computers and do not treat triclinic systems with periodicity. To our knowledge, no neighbour searching algorithms have been described in triclinic systems, with periodicity, running on an MIMD message passing computer.

## 1.1 Distributed neighbour searching

Designing a distributed neighbour searching algorithm and network, we were led by three wishes. First of all, the algorithm should be simple: no complex communication structure! Second, we want to use as much as possible the allowed roughness of the cutoff sphere. Third, we want to use T222's without external RAM as much as possible.

These wishes led to the following algorithm. Divide the primitive system in Q slices (Q is typically 100) of equal thickness (fig. 3). These slices should be in parallel with let us say the plane $\vec{A}, \vec{B}$. ($\vec{A}, \vec{B}, \vec{C}$ are the vectors spanning the triclinic shaped primitive system). The key idea in our neighbour searching strategy is to construct the cutoff sphere from circular discs, one disc per slice, and to run one search process per slice. Therefore it is necessary to sort particles over slices according to their position in the $\vec{C}$ direction. After a particle arrives at the slice where it belongs, its images no further than $R_{co}$ outside the primitive system, in the plane $\vec{A}, \vec{B}$, are created. At this instant, images are given the right extension, only one out of

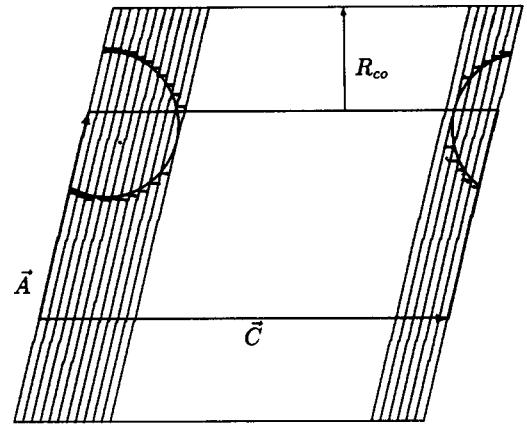



Figure 3: Primitive system, with slices and cutoff sphere, surrounded by image layer.

nine because only images in the plane $\vec{A}, \vec{B}$ are created. To make it possible to start processes in neighbouring slices, search processes should be connected in a linear way; the process in slice k should be able to start a process in slice k-1 or/and k+1. Periodicity in the C direction is taken care of by connecting processes in slice 1 to processes in slice Q. When going from slice 1 to slice Q or the other way around care should be taken to generate the right extension.

To generate the neighbour list of a particle, a search process is started in its own slice and two messages are sent to neighbouring slices, one to the left and one to the right. Every message initiates a new process and every process, if not out of the $R_{co}$ range, sends a 'start search process' message to the next outward slice. The partial neighbour lists generated by these processes have to be joined in one neighbour list, and this list has to be expanded and excluded.

### 1.1.1 Topology of the NS network

A transputer network, with a suitable topology to run the algorithm as described in the previous section, is given in figure 4. Incoming records have to be sorted and are sent to the linear array of search processors. This is done in layers 1..5. In layer 6 the actual searching is done. After that, partial neighbour lists, generated in layer 6, are directed to a processor in layer 11 by processors in layers 7..10. Which processor in layer 11 they are directed to, is determined by the indicator particle number of the partial neighbourlist.(Particle number modulo 6 will do.) In layer 11, partial neighbour lists are joined, expansion and exclusion is done, and complete neighbour lists are transmitted to the NBFP. Processors in layer 11 should have enough memory to

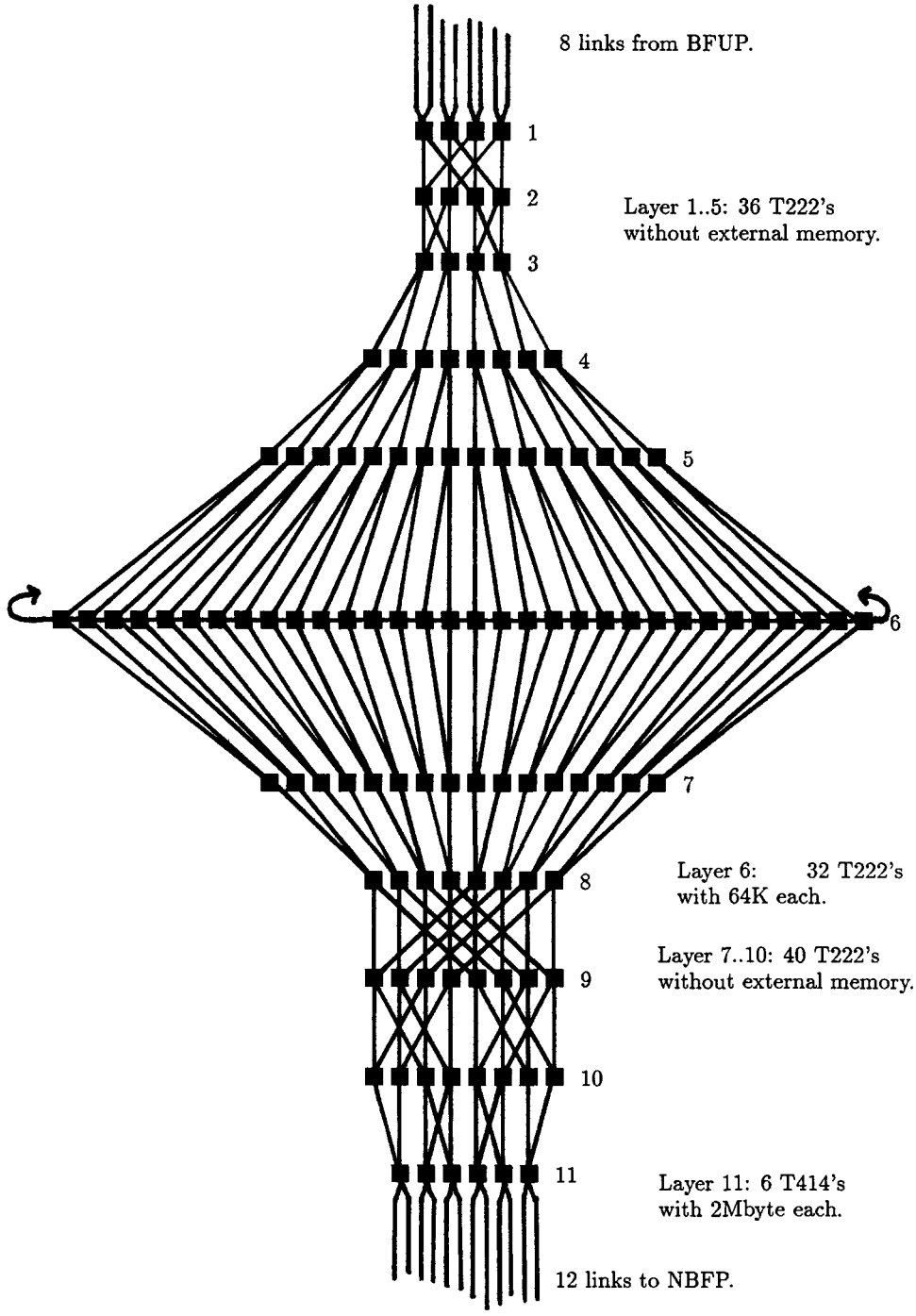



Figure 4: NS transputer network.

store copies of the neighbour lists transmitted to the NBFP. We think 2Mbyte/processor will do. Therefore in layer 11 not T222's but T414's should be used. (A T414 is a 32 bit transputer with four links, 2Kbyte on chip memory and an address space of 4Gbytes) In this way, in case the performance of the NS network is too low, we are slowed down only one out of M timesteps.

We think this is a nice NS topology because:

- Small scale tests indicate that a linear array of 32 processors in layer 6 will give us the required average speed of finding one particle within $R_{co}$ per 200 nsec.
- Many T222's without external RAM may be used; only in layer 6 and 11 are transputers with external RAM.
- By taking slices in the plane $\vec{A}, \vec{B}$, generating images can be done in the processor in which they belong; images do not have to be distributed over slices.
- Having generated a layer of image particles, one does not need to worry about periodicity; just search neighbours for every indicator particle in the primitive system.
- By wrapping around the linear array in layer 6, no images in the $\vec{C}$ directions have to be generated and stored, and the load balance of transputers in layer 6 is not disturbed by edge effects.
- By routing every neighbour list in layers 7..10 to its specific processors in layer 11 , we do not have to store in every transputer of layer 11 a copy of the exclusion list of every particle.

### 1.1.2 Some small feasibility studies

To get some feeling for the feasibility of this design we studied some literature and did a few small experiments.

**Blocking.** In layers 1..3 and 8..11 two omega networks are used to do sorting. Every node in these networks has two incoming and two outgoing links. A problem arises when two incoming packets in a node want to use the same outgoing link; that is called blocking. If we suppose on the incoming links of a node an infinite stream of packets, each with probability 1/2 to choose the left or right outgoing link, and if we suppose infinite buffers in the node then it is clear that the time loss due to blocking tend to zero. However, because a T222 transputer has only 4Kbyte of internal memory, we should find out how much buffering is needed to approach the infinite buffer behaviour. This question is answered by Dias and Jump in [10], where it is shown that a buffer size of 10 packets gives a throughput of 90% of the maximum throughput. In our case this means a buffer space of 2*10*8=160 bytes. (one buffer per outgoing link, 8 bytes per packet) A buffer of this size gives no problems in the 4Kbyte memory of the T222, so we do not need to worry about blocking.

**Load balance in layer 6.** Incoming packets are distributed over transputers in layer 6. If we suppose that P particles are distributed over the M.D. space with uniform probability, that these particles are distributed over T transputers as described in a previous section, and that P/T is sufficiently large (let's say $\geq$10), then the number of particles per transputer is a Gaussian distribution centered at P/T and a width of $\sqrt{P/T}$. For a very small M.D. system of 320 particles and T=32 this means 10$\pm$3.16 particles per transputer. Because the workload per transputer is roughly proportional to the squared number of particles on that transputer, an unbalance of $13.16^2/10^2 = 1.73$ may be expected, that is, the slowest transputer has finished its searching 73% later than the average transputer. Fortunately, a system consisting of 320 particles is unrealistic; our M.D. computer will handle systems of 3000 to 64000 particles. In this range, the unbalance will be 21% to 5%. We are willing to accept this unbalance, and therefore we will not modify the particle distribution algorithm in such a way that particles are distributed more evenly over transputers.

**Search process scheduling.** The hardware that accepts neigbourlist from the NS (so the non bonded force processor (NBFP)), expects neighbourlist items at a rate of one item per 200nsec. In layer 11 of the NS some buffering may be done to smoothen the stream of neighbourlists generated in layer 6, but because of the limited buffering capacity of layer 11, and to minimize the start up time of the NS, layer 6 should produce complete neighbourlists as soon as possible and with a constant speed of at least one item per 200nsec. Producing *complete* neighbourlists is necessary because otherwise in layer 11 many partial neighbourlists will accumulate, which will be completed in the last stage of the neighboursearching process. This uncoordinated way of neighboursearching causes a highly non continuous stream of neighbourlists between NS and NBFP.

In order to generate a constant stream of complete neighbourlists, the following scheme may be used. A distributed dispatcher should be implemented in layer 6, moving in a circular way from transputer to transputer in layer 6. At every transputer it checks to see if there are enough partial search processes in the ready queue. If so, the dispatcher moves to the next transputer. If not, a process is created, searching neighbours of a particle located at that transputer. This process is given a unique number: the previous number plus

| | searching with projection. | | |
|---|---|---|---|
| *Rco* | N=100 | N=400 | N=900 |
| 1 | 118 $\mu$sec | 201 $\mu$sec | 266 $\mu$sec |
| 2 | 44 | 74 | 104 |
| 4 | 20 | 32 | 45 |
| 8 | 11 | 17 | 23 |
| 15 | 11 | 12 | 14 |

| | naïve searching | | |
|---|---|---|---|
| *Rco* | N=100 | N=400 | N=900 |
| 1 | 235 $\mu$sec | 963 $\mu$sec | 2094 $\mu$sec |
| 2 | 54 | 201 | 446 |
| 4 | 16 | 48 | 105 |
| 8 | 6 | 15 | 29 |
| 15 | 5 | 7 | 11 |

$N$ is number of particles per slice.
Box length is $\sqrt{N}$.
Programmed in Occam; run on a T800.

Table 1: Time needed to find an in range pair.

one. When this process is distributed over neighbouring transputers, the unique number goes with every partial search process to these transputers. This was the process creation; now the process execution. At every transputer in layer 6, partial search processes should be executed in increasing process number order. In this way, partial search processes do not suffer from starvation; on the contrary, they are executed after at most R other partial search processes have been executed. (R is a fixed number, depending on the dispatcher algorithm and $R_{co}$.)

**Search strategies within a slice.** We tried different algorithms for searching within a slice: Bentley's K-D trees [8], naïve searching and projection. K-D trees only worked with an unrealistic number of particles per slice (over 2000) where a realistic number is 200 for a 20000 particle, 96 slice system. Naïve searching, that is calculating all particle distances, is inferior to projection in one direction when only a small portion of all possible pairs is found. However, doing naïve searching with $R_{co}$ = 15 and N = 400 on the average a time of 7 $\mu$sec was needed to find a pair of particles which are separated less than $R_{co}$(see table1). If layer 6 consists of 32 transputers then in this case every 7/32$\mu$sec $\approx$ 220 nsec a pair is found. This comes close to the 200 nsec as required in the specification. However, for other values of $R_{co}$ the search times must be improved. This can be done in three ways. One is to improve the algorithm for projection by sorting in both directions. This will increase the preprocess time but this time is two orders of magnitude smaller than the search time, so this will not matter much, but will decrease the time per pair. The second option is to use lower level languages than we have used, which will make programming harder but will increase the throughput. And thirdly the problem is scalable: using more transputers in layer 6 will decrease the time per pair. We hope that the combined effects of these improvements will give the speed of 200 nsec/pair over the whole $R_{co}$ range.

**Alternative network topologies.** To save on the number of transputers, we consider a topology without layers 1..5. We will then use the omega network in layers 8..11 to sort incoming as well as outgoing data. Apart from the decrease in the number T222's, this topology has the advantage of a higher bandwidth omega network for the incoming data stream; 16 instead of 8 links. However, we are not sure if the combined algorithms fit in the internal T222 memory.

Another possible change in the topology is the use of 48 instead of 32 transputers in layer 6. This may be necessary when 32 transputers do not have enough performance to find every 200nsec. an in range pair of particles. Using 48 instead of 32 transputers in layer 6 does hardly influence other parts of the network; three instead of two links should be implemented between the transputers in layers 5,6 and 7,6.

**Concluding.** We will continue small scale experiments, and probably will be funded in the second half of 1990 to construct the whole NS as well as the NBFP and BFUP.

**Acknowledgements** We wish to thank R. Moddemeijer for proofreading this paper and A. Sijbers for helping with the transputer software.

## 1.4 Literature

[1] A.F. Bakker, C. Bruin, F. van Dieren and H.J. Hilhorst, Molecular dynamics of 16000 Lennard-Jones particles Phys. Lett. 93A, 67 (1982).
[2] L. Verlet, Phys. Rev., 159, 98 (1967).
[3] J.L.. Finney, J. Comput. Phys., 28, 92 (1978).
[4] W.B. Street, D.J. Tildesley and G. Saville, Mol. Phys., 35, 639 (1978).
[5] R.W. Hockney, S.P. Goel and J.W. Eastwood, J. Comput. Phys., 14, 148 (1974).
[6] R.W. Hockney and J.W. Eastwood, Computer simulation using particles, McGraw-Hill, New York, 1981.
[7] B. Quentrec and C. Brot, J. Comput. Phys., 13, 430 (1973).
[8] J.L. Bentley, Multidimensional binary search trees used for associative searching, Comm. of the ACM, 18 ,9 (September, 1975).
[9] W.F. van Gunsteren, H.J.C. Berendsen, F. Colonna, D. Perahia, J.P. Hollenberg, D. Lellouch, On searching neighbours in computer simulations of macromolecular systems, J. Comput. Chem. 5, 3, 272-279, (1984).
[10] Daniel M. Dias and J. Robert Jump, Analysis and simulation of buffered delta networks, IEEE transactions on computers, vol. C-30, no. 4 April 1981.